3810 Review Session

Spring 2024

Disk Basics

- Disk access remains very slow – mechanical head that has to move to the correct “ring” of data – order of milli-seconds – high enough that a context-switch is best
- Focus on other metrics, especially reliability
- A sector on the disk is associated with a cyclic redundancy code (CRC) – a hash that tells us if the read data is correct or not – it is simply an error detector, not an error corrector
- To correct the error, RAID is commonly used
- Reliability measures continuous service accomplishment and is usually expressed as mean time to failure (MTTF)
- Availability is measured as MTTF/(MTTF+MTTRecovery)

RAID

- RAID 0: no redundancy
- RAID 1: mirroring
- RAID 2 and 6: memory-style ECC and rarely deployed
- RAID 3: bit-interleaved, lower cost, but no query-level parallelism
- RAID 4: block-interleaved, lower cost, query-level parallelism, but write bottleneck
- RAID 5: block-interleaved, lower cost, query-level parallelism, write parallelism
- Parity and XOR!

Pipeline Performance

Unpipelined processor

CPI:

Clock speed:

Throughput:

Pipelined processor

CPI:

Clock speed:

Throughput:

Circuit Assumptions

Length of full circuit:

Length of each stage:

No hazards

Data Hazards

No Bypassing
(for the 5-stage pipeline)
Point of production: always RW middle
Point of consumption: always D/R middle

```
                                 * PoP
I1  add:   IF   DR    AL    DM    RW
I2  add:       IF     DR    DR    DR    AL  DM  RW
                                 * PoC
```

Bypassing

Point of production:
- add, sub, etc.: end of ALU
- lw: end of DM

Point of consumption:
- add, sub, lw: start of ALU
- sw $1, 8($2): start of ALU for $2,
  start of DM for $1

```
                                 * PoP
I1  add:   IF   DR    AL    DM    RW
I2  add:       IF     DR    AL    DM  RW
                                 * PoC
```

Control Hazards

Assumptions

100 instructions
20 branches
14 Not-Taken, 6 Taken
Branch resolved in 6th cycle (penalty of 5)

Approach 1: Panic and wait

Approach 2: Fetch-next-instr

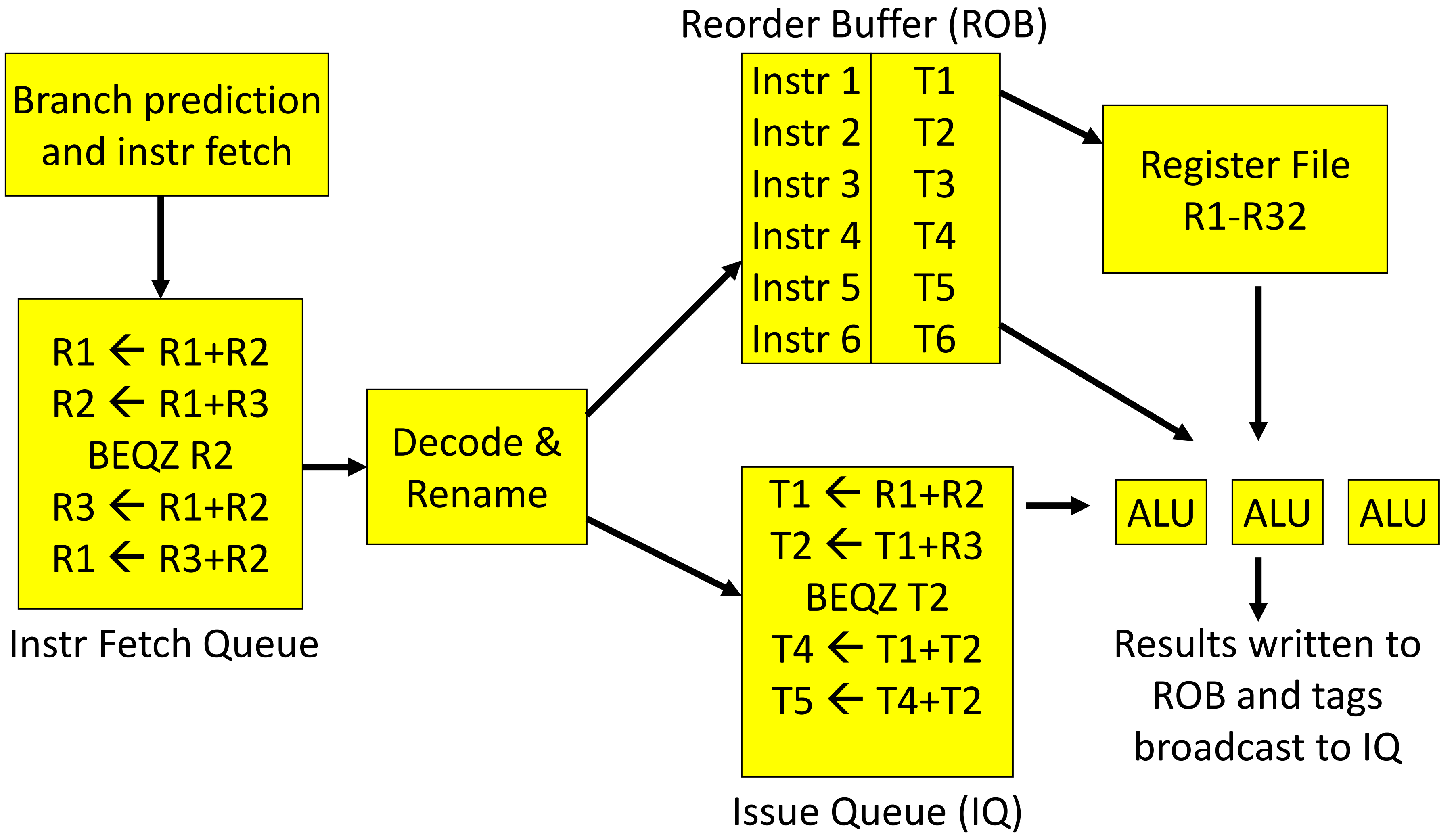
Approach 3: Branch Delay Slot

Option A: always useful
Option B: useful when the branch goes along common fork
Option C: useful when the branch goes along uncommon fork
Option D: no-op, always non-useful

Option A
Branch
Slot
NTaken Taken
Option B Option C

Approach 4: Branch predictor

Accuracy of 90%



Out of Order Processor

Assumptions

1000 instructions, 1000 cycles, no stalls with L1 hits

# loads/stores:

% of loads/stores that show up at L2:

% of loads/stores that show up at L3:

% of loads/stores that show up at mem:

L2 acc = 10 cyc,   L3 acc = 25 cyc,   mem acc = 200 cyc

Cache Latency

Assumptions

512KB cache, 8-way set-associative, 64-byte blocks, 32-bit addresses

Data array size = #sets x #ways x blocksize
Tag array size = #sets x #ways x tagsize
Offset bits = log(blocksize)
Index bits = log(#sets)
Tag bits + index bits + offset bits = addresswidth

Cache Size

Cache Hits/Misses

Assumptions

16 sets, 1 way, 32-byte blocks

Access pattern: 4 40 400 480 512 520 1032 1540

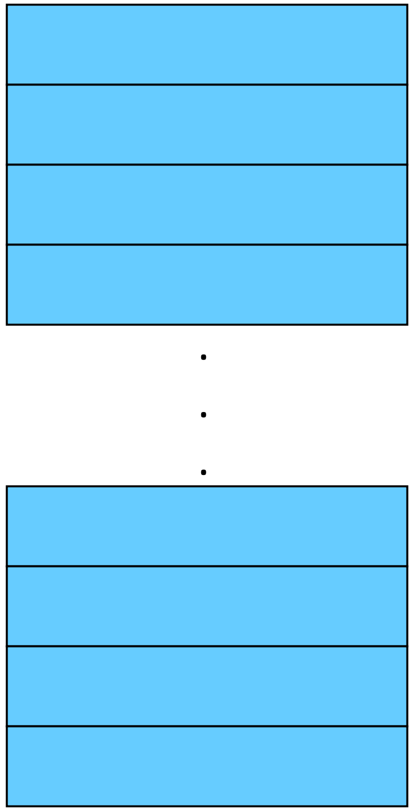
Offset = address % 32 (address modulo 32, extract last 5)
Index = address/32 % 16 (shift right by 5, extract last 4)
Tag = address/512 (shift address right by 9)

32-bit address

	23 bits tag	4 bits index	5 bits offset	H/M	Evicted address
4:	0	0	4	M	Inv
40:	0	1	8	M	Inv
400:	0	12	16	M	Inv
480:	0	15	0	M	Inv
512:	1	0	0	M	0
520:	1	0	8	H	-
1032:	2	0	8	M	512
1540:	3	0	4	M	1024

Example 0b

Show how the following addresses map to the cache and yield hits or misses.
The cache is direct-mapped, has 16 sets, and a 64-byte block size.
Addresses: 8, 96, 32, 480, 976, 1040, 1096



Offset = address % 64 (address modulo 64, extract last 6)
Index = address/64 % 16 (shift right by 6, extract last 4)
Tag = address/1024 (shift address right by 10)

32-bit address

	22 bits tag	4 bits index	6 bits offset	
8:	0	0	8	M
96:	0	1	32	M
32:	0	0	32	H
480:	0	7	32	M
976:	0	15	16	M
1040:	1	0	16	M
1096:	1	1	8	M

Consider a 3-processor multiprocessor connected with a shared bus that has the following properties:

(i) centralized shared memory accessible with the bus, (ii) snooping-based MSI cache coherence protocol, (iii) write-invalidate policy. Also assume that the caches have a writeback policy. Initially, the caches all have invalid data. The processors issue the following three requests, one after the other. Similar to slide 17 of lecture 25, fill in the following table to indicate what happens for every request. Also indicate if/when memory writeback is performed. (8 points)

P2: Read X
P1: Read X
P2: Write X
P3: Read X

Request	Cache Hit/Miss	Request on the bus	Who responds	State in Cache 1	State in Cache 2	State in Cache 3	State in Cache 4
				Inv	Inv	Inv	Inv
P2: Rd X							
P1: Rd X							
P2: Wr X							
P3: Rd X							

Questions to ask yourself:

How does Meltdown work?

How does Spectre work?

How can you force a footprint? (the relevant code sequence)

How can you examine footprints? (exploiting the side channel)

How can you defend against these attacks?

Questions to ask yourself:
What does the programmer/compiler deal with?
What does the OS deal with?
How is translation done efficiently?

Virtual Memory

Questions to ask yourself:

Why do multiprocs need to deal with prog. models, coherence, synchronization, consistency?

What are race conditions?

What is an example synchronization primitive and how is it implemented?

What consistency model is assumed by a programmer?

Why is it slow?

How do I make life easier for the programmer and provide high performance?

Synchronization, Consistency

Questions to ask yourself:
What are the central philosophies in a GPU?
In what ways does the GPU design differ from a CPU?
What are the different ways that disks provide high reliability?
Can you explain how parity is used to recover lost data?

GPUs, Disks